# A Design-Diversity Based Fault-Tolerant COTS Avionics Bus Network

Savio N. Chau Joseph Smith
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

Ann T. Tai
IA Tech, Inc.
10501 Kinnard Avenue
Los Angeles, CA 90024

**Abstract**

*This paper describes a COTS bus network architecture consisted of the IEEE 1394 and SpaceWire buses. This architecture is based on the multi-level fault tolerance design methodology proposed by Chau[1] but has much less overhead than the original IEEE 1394/I2C implementation. The simplifications are brought about by the topological flexibility and high performance of the SpaceWire. The connections in SpaceWire can form a connected graph that embeds multiple spanning trees. This allows the IEEE 1394 bus to select a different tree topology when a fault occurs. It also has sufficient performance to stand in for IEEE 1394 bus during fault recovery, so that a backup IEEE 1394 bus is not required. These two buses are very compatible at the physical level and therefore can easily be combined. Analysis of the effectiveness of the IEEE 1394/SpaceWire architecture shows that it has the same fault tolerance capability as the IEEE 1394/I2C architecture with much simpler connectivity.*

## 1. Introduction

In recent years, commercial-off-the-shelf (COTS) products have found many applications in space exploration. The benefit of COTS is that low cost hardware and software products are widely available in the commercial market. By using COTS through out the system, both the development cost as well as the recurring cost of the system can be reduced. On the other hand, COTS are not specifically developed for highly reliable applications such as long-life deep-space missions. Therefore, developing low-cost COTS based systems with the same level of high reliability as the traditional custom-designed space system is a major challenge. To meet this challenge, the X2000 Project at the Jet Propulsion Laboratory (JPL) has developed a multi-level fault-tolerance design methodology to improve the reliability of COTS based systems [1]. The methodology contains four layers of fault tolerance design: Native Fault Containment, Enhanced Fault Containment, Design-Diversit, and System Level Redundancy.

This methodology has been applied to the design of a fault tolerant avionics bus architecture consisted of an IEEE 1394 bus and an I2C bus. A testbed has been developed for the bus architecture [3] and the ASIC components are being developed [4]. A fault injection campaign on the testbed has been planned in the next two years. The theoretical cost effectiveness of the methodology has also been analyzed [2].

One reason that the IEEE 1394 and I2C buses are selected in this architecture is because of the diversity of their topologies. While this choice of buses is capable of achieving high reliability, the performance mismatch between the IEEE 1394 (100 Mbps) and the I2C (100 Kbps) buses renders the I2C Bus ineffective as a backup bus to the IEEE 1394 bus. Hence, it becomes necessary to have the system level redundancy, in which both the IEEE 1394 and the I2C buses are duplicated.

In this paper, a much more capable bus, the SpaceWire, will be introduced to replace the I2C bus. The SpaceWire bus has bandwidth comparable to the IEEE bus. With such capable backup bus, the system level redundancy will become optional. This is particularly beneficial for smaller spacecraft that are constrained by mass and volume.

## 2. Multi-level fault tolerance design methodology for COTS

COTS are not developed with the same level of rigorous fault tolerance in mind. Therefore, they usually have many major shortcomings in fault tolerance. For examples, the popular VME bus does not even have the parity bit to check the data and address [11] and the IEEE 1394 bus (cable implementation) adopts a tree topology in which a single node or link failure will partition the bus. These fundamental weakness will hinder rigorous enforcement of fault containment. Worse yet, it is almost impossible to make modifications to COTS in general. There are two reasons. First, the suppliers of COTS products have no interest to change their design, add any overhead, or sacrifice their performance for a narrow market of high reliability applications. Second, any modification will render the COTS incompatible with commercial test equipment or software, and therefore diminish the economic benefits of COTS drastically. Therefore, it is obvious that fault tolerance cannot easily be achieved by a single layer of fault containment regions that contains COTS.

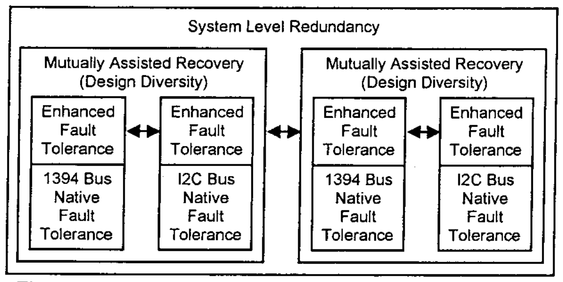
To overcome these shorcomings, the COTS-based bus architecture of the X2000 has employed a multi-level fault protection methodology to achieve high reliability. The methodology is consisted of several levels of protections:

*Level 1: Native Fault Containment* – most of COTS bus standards have some limited fault detection capabilities. These capabilities should be exploited as the first line of defense.

*Level 2: Enhanced Fault Containment* – addition layer of hardware or software can be used to enhance the fault detection, isolation, and recovery capabilities of the native fault containment region. Examples are watchdog timer or additional layer of error checking code. It is important to ensure that the added fault tolerance mechanisms will not affect the basic COTS functions. This level is also the most convenient level to implement provisions for fault injections.

*Level 3: Fault Protection by Component Level Design-Diversity* – many COTS have fundamental fault tolerance weakness that cannot simply be removed by enhancing the native fault protection mechanisms. These weakness usually are related to single points of failures. For example, branch node failure in a buses with tree toplogy is a single point failure, which can be compensated with another bus with different topology such as multi-drop, ring etc..

*Level 4: Fault Protection by System Level Redundancy* – for system that has very high reliabiity requirements, the fault tolerance mechanisms in levels 1 – 3 can be replicated to further enhance system level reliability. The redundant fault tolerance mechanisms can be either in ready or dormant states, depending on the recovery time and other system requirments. If they are in the eady state, voting or comparison can reduce the fault detection and/or recovery time. These four levels of fault tolerance are depicted in Figure 1.



System Level Redundancy
Mutually Assisted Recovery (Design Diversity)
Enhanced Fault Tolerance
Enhanced Fault Tolerance
1394 Bus Native Fault Tolerance
I2C Bus Native Fault Tolerance
Mutually Assisted Recovery (Design Diversity)
Enhanced Fault Tolerance
Enhanced Fault Tolerance
1394 Bus Native Fault Tolerance
I2C Bus Native Fault Tolerance

Figure 1: COTS bus architecture fault-tolerance strategy

## 3. The COTS-based X2000 fault tolerant bus architecture

The X2000 Project at Jet Propulsion Laboratory is a technology program to develop highly integrated, reliable, and high performance micro-avionics systems for future deep space missions. The avionics architecture that the X2000 Project has developed is a distributed, symmetric system of multiple computing nodes connected by a fault tolerant bus network. Standard COTS interfaces are used throughout the architecture for cost, compatibility, and scalability reasons. In particular, the Peripheral Component Interface (PCI) bus is used as the local computer bus; the IEEE 1394 bus and the $I^2C$ bus are the "system" buses. Within each node, there is also a separate "subsystem" $I^2C$ bus for sensors and instruments control (see Figure 2).

The implementation of the multi-level fault tolerance methodology in the bus architecture is as follows:

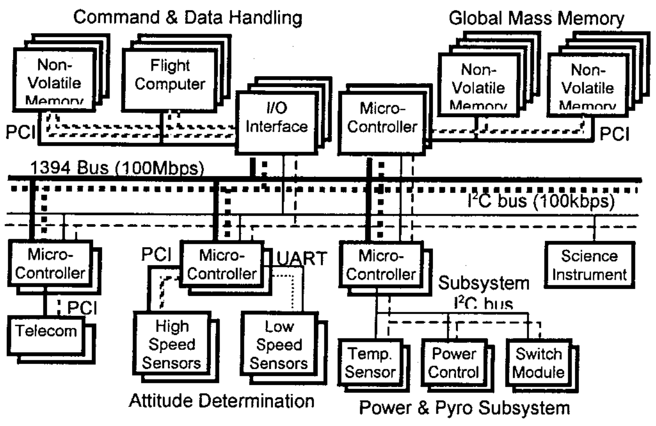


Figure 2: X2000 Avionics System Architecture

### 3.1 Native fault containment regions

This level exploits the basic fault detection mechanisms of the IEEE 1394 and $I^2C$ buses. A highlight of these mechanisms are as follows.

### 3.1.1 IEEE 1394 bus fault tolerance mechanisms

The IEEE 1394 bus is the artery of the system and is capable to transfer data at 100, 200, or 400 Mbps. Its protocol has two types of transactions. The isochronous transaction is designed for on-time delivery. The isochronous transaction is delimited by an *isochronous cycle start* packet every 125 μs. Every isochronous node is entitled to send data according to its bandwidth allocation within the isochronous cycle. It is inherently multi-cast and thus does not require acknowledgement from the nodes receiving the messages. The asynchronous transaction uses node-to-node transfers. It is designed for reliable delivery and therefore requires the receiving node to acknowledge the completion of data transfer. The IEEE 1394 bus has two implementations, cable and backplane. The cable implementation has a tree topology while the backplane implementation is a multi-drop bus. The X2000 Project has adopted the cable implementation because of its much wider industrial supports.

The 1394 bus standard has many built-in fault detection mechanisms. Examples of these mechanisms includes:

1. Data and packet header CRCs for both isochronous and asynchronous transactions
2. Acknowledgment packets include error code to indicate if the message has been successfully delivered in asynchronous transactions
3. Parity bit to protect acknowledgment packets
4. Response Packets include error code to indicate if the requested action has been completed successfully in asynchronous transactions
5. Timeout conditions: response timeout for split transaction, arbitration timeout, acknowledgment timeout etc.

A very import feature in the latest version of the IEEE 1394 standard (IEEE 1394a [10]) is the capability to enable or disable individual ports (a port is the physical interface to a link). With this feature, every node in the bus can disable a link connected to a failed node and enable a backup link to bypass the failed node. This feature is the basis of the IEEE 1394 bus recovery in this bus architecture.

Another feature in the IEEE 1394 standard is the keep-alive of the physical layer with cable power. This feature allows the link layer hardware and the host processor to be powered off without affecting the capability of the physical layer to pass on messages. This is useful for insolating a failed processor during fault recovery.

### 3.1.2 I$^2$C bus fault detection mechanisms

The I$^2$C bus is a simple bus with a data rate of 100 kbps. It has a more traditional multi-drop topology. The I$^2$C bus has two open-collector signal lines: a data line (SDA) and a clock line (SCL). Both signal lines are normally pulled high. When a bus transaction begins, the SDA line is pulled down before the SCL line. This constitutes a start condition. Then the address bits will follow, which is followed by a read/write bit and then an acknowledgment bit. The target node can acknowledge the receipt of the data by holding down the acknowledgment bit. After that, eight bits of data can be sent followed by another acknowledgment bit. Data can be sent repeatedly until a stop condition occurs, in which the source node signals the end of transaction by a low-to-high transition on the SDA line while holding the SCL line high.

The I$^2$C uses collision avoidance to resolve conflicts between master nodes contending for the bus. If two or more masters try to send data to the bus, the node producing a 'one' bit will lose arbitration to the node producing a 'zero' bit. The clock signals during arbitration are a synchronised combination of the clocks generated by the masters using the wired-AND connection to the SCL line.

The only fault detection mechanism of the I$^2$C bus is the acknowledgment bit that follows every data byte. When a node (master) sends data to another node (slave), and if the slave node is able to receive the data, it has to acknowledge the transaction by pulling the data line (SDA) to low. If the slave node fails to acknowledge, the master node will issue a stop condition to abort the transaction. Similar situation can happen when the master node requests data from a slave node. If the master fails to acknowledge after receiving data from the slave, the slave will stop sending data. Subsequently, the master node can issue a stop condition to terminate the transaction if it is still functional.

## 3.2 Enhanced fault containment regions

Several mechanisms are added to enhance the fault detection and recovery capability of the IEEE 1394 and $I^2C$ buses. They are discribed in the following.

### 3.2.1 Enhanced Fault Tolerance Mechanisms for IEEE 1394 Bus

***Heartbeat and Polling:*** The X2000 architectural design enhances the fault detection capability of the 1394 bus with heartbeat and polling. Non-root nodes can use heartbeat to detect connection or branch node failures upstream, while the root node can use polling to detect fail-silent leaf nodes. Heartbeat can be implemented by the cycle start message, which is sent by the root node every 125 microseconds (average). Polling has to be implemented by additional polling messages that the root node sends to individual nodes. Additional watchdog timers have to be used to detect heartbeat or polling failures.

***Backup Connections:*** In addition to the connections in the tree topology, backup connections can be added so that the bus can be configured upon node or link failures to recover the topology. Cautions have to be used, however, because the backup connections can create loops, which are prohibited by the IEEE 1394 bus standard. Fortunately, IEEE 1394a, the updated version of the standard, allows connection ports to be disabled. Hence, the backup connections can be disabled during normal operations but activated as required during fault recovery. Additional software need to control the enable and disable of the connection ports.

### 3.2.2 Enhanced Fault Tolerance Mechanisms for $I^2C$ Bus

***Protocol Enhancement:*** A layer of protocol is added to the $I^2C$ bus. This protocol includes a byte count after the address and two CRC bytes after the data. X2000 design also utilizes especial hardware messages commands to control critical functions. For these messages, command is sent followed by its complement to provide one more layer of protection.

***Fail Silence:*** Some failure modes in the I2C bus can corrupt the entire bus. Examples include output drivers shorted to ground or babbling nodes. The X2000 Project has developed a fail silence protocol to handle these failures. Every node periodically sends a "fail silence" message to itself. If the bus is corrupted by a babbling node or a shorted bus driver, the fail silence message will not be received in time and a fail-silence timer in the node will time out. Consequently, the bus transmitters of the node will be disabled to inhibit any transmission of messages, but bus receiver is still enabled so that it can receive commands for fault recovery later on. Eventually, all the nodes including the failed node will be disable, so that the bus is free of fault again. After a node has disabled its bus transmitter, it will start an "unmute" timer that will re-enable its bus transmitter when the timer times out. The value of the unmute timer is assigned randomly. In time, one of the nodes will unmute itself first. This node will send a message to re-enable the other nodes individually. If the bus fails again after a node enabled, then the node will be identified as the failed node and will not be enabled again.

## 3.3 Fault protection by design diversity

By working together, the IEEE 1394 and $I^2C$ buses can isolate and recover from many faults that might not be possible if each bus is working alone. .

***IEEE 1394 Node or Connection Failures***: When a branch node or one of its connections fails, messages will not be able to pass through the node and the bus is partitioned into two or three segments. The nodes in each segment will not be able to communicate with the nodes in another segment. Consequently, the root node will not be able to command the nodes in the other segments to reconfigure in order to recover from the fault. In this situation, the $I^2C$ bus can facilitate the communication among all the nodes. The root node can first send messages to interrogate the health of all nodes in every segment through the $I^2C$ bus. After receiving the interrogating message, each node will perform a thorough test including its ports and connections, and then report the findings to the root node through the I2C bus. After the location of the fault is identified, the root node will send $I^2C$ messages to the nodes around the faulty location to command them to reconfigure their ports to bypass the fault. If the root node is the one that fails, then one of its children nodes can take over the control of the recovery process.

***I2C Bus Failures:*** For any failure modes in the I2C bus that are not covered by the fail-silence protocol, nodes can interrogate the health of each other through the IEEE 1394 bus and disable the nodes that cause the I2C bus failure.

## 3.4 Fault protection by system level redundancy

The COTS bus set is duplicated to provide system level of fault protection. When a fault is detected in the primary bus set, simple recovery procedures such as retry and bus reset will first be attempted. If the simple procedures cannot correct the problem, then the backup set of buses will be activated and the system operations will be transferred to the backup bus. At this point, the system can have more time to diagnose the failed bus set and remove the faulty node or connections using the techniques mentioned above. The repaired bus set will then become the backup

To further enhance the effectiveness of the system level redundancy, the IEEE 1394 bus in the backup COTS bus set connects the nodes in such a way that any branch node in primary bus set is a leaf node in the backup bus set and vice versa. This would prevent any node to become a branch node for both buses. Hence, a failed node can only partition the bus in which it is a branch node. For the other bus, the failure only represents the loss of a leaf node, but the main body of the tree structure is not affected. An example that follows this rule is the "stack-tree" topology shown in Figure 3. More detailed description of the X2000 fault tolerant bus architecture can be found in [#][#]

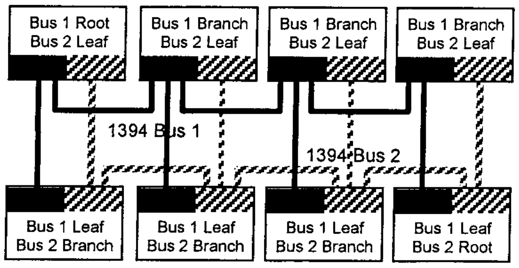


Figure 3: Stack-tree topology of IEEE 1394 bus

## 4. Issues about the current X2000 fault tolerant bus architecture

Although the current X2000 fault tolerance bus architecture is very robust, the amount of redundancy might be more complicate than necessary. The reason is that the perfomance of the I2C bus is much lower than the IEEE1394 bus due to its simplicity. This is an advantage for some systems such as plantary surface landers and rovers. These sysstems usually have relatively short mission life and do not have the power or volume to support two redundant IEEE 1394 buses. On the other hand, for long life missions, the performance disparity between the IEEE 1394 bus and the I2C bus necessitates a redundant set of IEEE 1394/I2C buses. This is because when the IEEE 1394 bus fails, the I2C is not capable to substitute for the IEEE 1394 bus. Therefore, the backup IEEE 1394 bus is required to take over the workload while the I2C bus is assisting the original IEEE bus to recover. For this reason, if the I2C bus can be replaced by a more capable bus such that the alternate bus has sufficient performance to temporary replace the IEEE 1394 bus and meanwhile supporting the IEEE 1394 bus fault recovery, then the system level redundancy will not be necessary except for those missions that requires extremely high reliability. A viable candidate of this alternative bus is the SpaceWire, which is described in the following.

## 5. SpaceWire

The SpaceWire standard is an adaptation of the IEEE 1355 bus by the European Space Agency (ESA) for space applications[#]. It is a full-duplex, serial, point-to-point network. Six levels of protocols are defined in this standard: physical, signal, character, exchange, packet, and network.

The physical level of the standard defines the cable and connector formats and the signal level defines the data signaling. The signal level of the SpaceWire is almost identical to the IEEE 1394-1995 and IEEE 1394a standards. It also has the Data and Strobe signals and both signals are carried by twisted wire pairs. Its bus driver also complies with the Low Voltage Differential Signaling (LVDS) standard IEEE 1596.3. However, unlike the IEEE 1394 bus, the SpaceWire only specifies the minimum data rate as 2 Mbps but does not specify the maximum data rate. However, it is expected that its highest data rates are comparable to the IEEE 1394. In fact, the jitter and skew budgets for 100 Mbps, 200 Mbps, and 400 Mbps are provided in the standard. Another difference between SpaceWire and IEEE 1394 at the signal level is directionality. While the twisted wire pairs in IEEE 1394 carry signals in both directions, the twisted wire pairs in SpaceWire is unidirectional. Therefore, to achieve full duplex, four twisted wire pairs are required in Spacewire, two pairs for the Data and two pairs for the Strobe signals.

The character level protocol defines the character format There are two types of characters in SpaceWire: data and control (Figure 4). A control flag in each character distinguishes the two types of characters; 0 indicates a data character and 1 indicates a control character. A data character contains a parity bit, a data control flag, and eight bits of data. There are two types of control characters: link character (L-Char) and normal characters (N-Char). L-Chars are used in the exchange level and are not passed onto the packet level. The L-Chars include Flow Control Token (FCT), Escape (ESC), NULL, and Time Code. N-Chars are passed on to the packet level to indicate the end of packet. The Normal End of Packet (EOP) and Error End of Packet (EEP) are N-Chars. There is protocol definition equivalent to the character level defined in IEEE 1394.

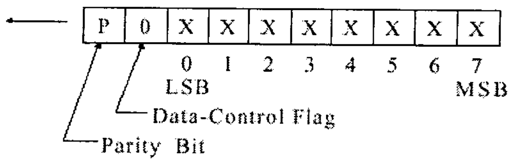


Figure 4: SpaceWire Character

The exchange level is responsible for making connection across a link and managing the flow of data across the link. The exchange level provides the following services: initialization of connection, flow control, detection of disconnect error, detection of parity error, and link error recovery. When a node attempts to initiate a connection with another node, it sends a NULL character to the destination node, waits to receive a NULL character, sends a FCT, and waits to receive a FCT. Full connection is established if the requesting node has received one or more NULL characters followed by the FCT character. The FCT character is used for flow control. By sending an FCT, the requesting node indicates to the destination node that it has space in its receive buffer for eight more characters. The NULL character is used to maintain the connectivity between two nodes.

In order to remain connected, the nodes at both ends of a link want have to send NULL characters continuously when they do not have data to send. When a disconnect error or parity error is detected, the nodes at both end stop transmitting, wait for 19.2 □sec, and then go through the NULL/FCT handshaking again to attempt to re-establish the connection.

The packet level defines how data is encapsulated in packets for transfer from source to destination. The format of a packet includes a destination address section, a cargo section, and an end of packet marker. The destination address section contains either the identity of the destination node or the path that the packet will take to get to the destination node. The cargo is the data to be transferred from the source to the destination. The end of packet marker is either an EOP or EEP character to indicate the end of packet.

The network level defines what a SpaceWire network is, describes the components that make up the SpaceWire network, explains how packets are transferred across the network and details the manner in which the SpaceWire network recovers from errors. A SpaceWire Network is made up of a number of nodes interconnected by routing switches. SpaceWire nodes may be directly connected together or using routing switches. A routing switch has several link interfaces connected together by a switch matrix that allows any link input to pass the packets that it receives on to any link output for re-transmission. An example SpaceWire network is shown in Figure 5.

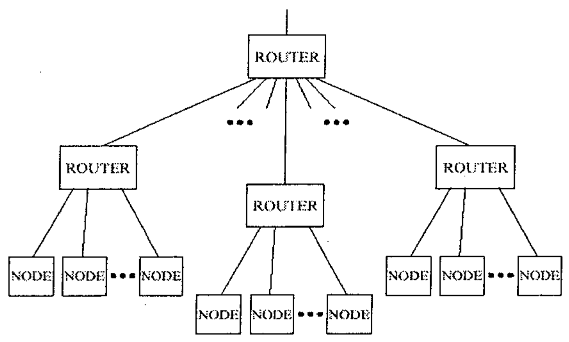


Figure 5: SpaceWire Network

Packets are routed through SpaceWire network using wormhole routing. Each packet contains a header that holds the destination node address either as the route through the network or as the identity of the destination node. As soon as the header for a packet is received, the routing switch determines which output port the packet must be routed to by checking the destination address. When more nodes are required, several switches can be cascaded to form larger networks. To arrive at its required destination, a packet may have to travel through several switches. The addresses of the switches along the route are specified as a series of headers in the destination address section of the packet. The leading header of the series indicates which is the next routing switch. As the packet goes through the switch, the leading header is stripped off, so that the next header is exposed and becomes the leading header. All the headers will be stripped off when the packet arrives the destination.

## 6. Comparison of the SpaceWire and the IEEE 1394 Buses

At the physical layer, the IEEE 1394 and the SpaceWire has comparable bandwidth. The major shortcoming of the IEEE 1394 bus is the tree topology since any branch node failure will partition the bus. On the contrary, the SpaceWire has a very flexible topology because the routing switches can support almost any connectivity. Another shortcoming of the IEEE 1394 bus is the complexity. The SpaceWire has a much simpler protocol and therefore the implementation is much less complicate.

On the other hand, the simpler protocol also means the SpaceWire has less capabilities at the link and transaction layers. For instance, broadcast and multi-cast can easily be implemented by the isochronous transaction in IEEE 1394 bus but are not supported by the SpaceWire. Broadcast and multi-cast have many useful applications such as time synchronization among subsystems or replicating messages for redundant subsystems.

Also, in IEEE 1394 bus, all nodes that have bandwidth allocation are entitled to transmit isochronous packets in every isochronous cycle. This ensures that a single node will not occupy the bus for a long time. This is particularly important for control loops because they are sensitive to variations of bus latency. On the contrary, when a connection is made between two nodes in the SpaceWire bus, they can continue the dialogue indefinitely and thus blocking other nodes from communication with either one of them. Hence, in principle, the bus latency in SpaceWire is more difficult to control, unless the system has only one controlling master node that can schedule the timing and duration of each transaction.

Therefore, the SpaceWire is a less capable bus than the IEEE 1394 bus for many space applications. It is more suitable for master-slave architecture while the IEEE 1394 is more suitable for distributed architecture. However, due to its high bandwidth and flexible topology, it is an ideal backup bus to support the recovery of the IEEE 1394 bus.

## 7. Design Diversity with the SpaceWire and the IEEE 1394 Buses

As it is shown in the previous section, the IEEE 1394 and the SpaceWire are very much complementary to each other. Therefore, it is possible that these two buses can be combined to provide a fault tolerant and yet very high performance bus network. Such bus network is described in the following.

### 7.1 Physical Level Compatibility Between the SpaceWire and IEEE 1394 Buses

At the physical level, the two buses share many similarities. First, the buses are electrically compatible because both of them use the IEEE 1569 Low Level Differential Signal (LVDS) bus drivers and twisted wire pairs. Second, both buses use the same data/strobe signaling.

The main difference between the two buses at this level is the directionality of the signals. In IEEE 1394, the signals in the twisted wire pairs are bi-directional. In fact, the conflicts of bus drivers on both ends of a connection are actually used as part of the protocol (e.g., arbitration request, arbitration grant etc.). On the other hand, the SpaceWire is a full duplex bus and thus requires two unidirectional connections for each signal. As a result, the IEEE 1394 bus cable has 6 wires: 2 pairs of twisted wires used for the data and strobe signals, 1 power, and 1 ground (An option in the standard has only 4 wires without power and ground). On the other hand, the SpaceWire cable has 8 wires: 2 pairs of twisted wires used for the data and strobe signals in one direction and 2 pairs for the opposite direction.

Regardless of their differences, these two buses can be physically combined as shown in figure 6. First, the routing switch of the SpaceWire is embedded in each node so that all components in the bus are identical. This will simplify fault recovery since it would not be necessary to have separate fault isolation and recovery techniques for nodes and routing switches. Second, selectors that are commendable by the processor are used to select which bus to be transmitted at each output.

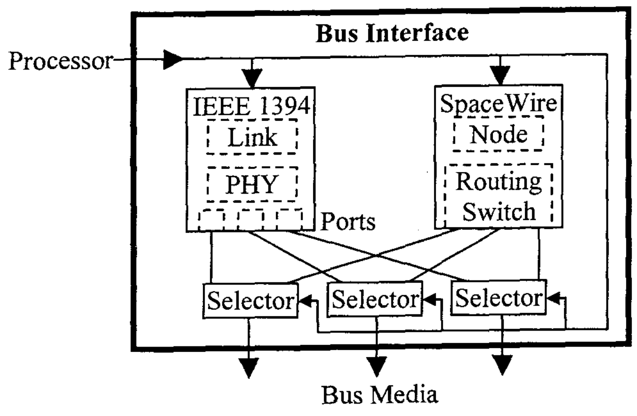


Figure 6: IEEE 1394/SpaceWire Combined Interface

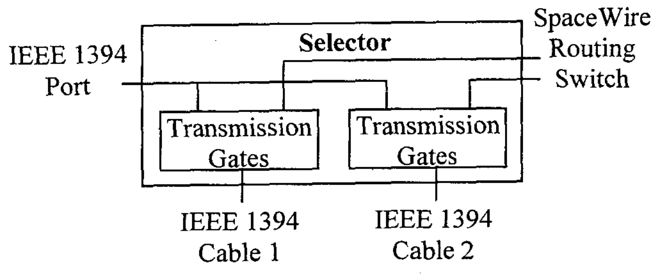


Figure 7: Selector Design

The design of the selector is shown in Figure 7. The output of the selector has two IEEE 1394 cables. If the IEEE bus is selected, it can use either one of these cables for connection. Otherwise, if the SpaceWire is selected, then one of the cables is used for transmitting signals in one direction while the other cable is used for transmitting signals in the opposite direction. The key to the selector design is that the transmission gates have to be passive and bi-directional, so that the electrical interface of either bus will not be affected.

Once a connection is selected for the IEEE 1394, then it will not be available to the SpaceWire. On the other hand, after the ports have been selected for the IEEE 1394, then either bus can perform its operations totally independent of the other bus. Therefore, there is no compatibility issue between the buses above the physical level.

## 7.2 Fault Tolerance Strategy with the Combined SpaceWire and IEEE 1394 Buses

With the capability to select bus in each system node, a highly fault tolerant and high-performance bus network can be developed. (Note: to avoid confusion, a system node refers to a unit that includes a processor, memories, and bus interfaces, while a SpaceWire node or IEEE 1394 node only refers to the part of the bus interface related to that bus.)

Since the SpaceWire has a much simpler bus initialization process and does not have the topological restriction like the IEEE 1394 bus, it is used as the default bus at the system start up. After the system is stabilized, a spanning tree will be selected from the connected graph formed by the connections of the SpaceWire. The process of selecting the spanning tree is as follows.

At system start up, one of the nodes will become the *network builder.* The *network builder* sends a *network building message* to all of its neighbors to request them to become its children. In the mean time, the *network builder* and its children nodes will change the edges (i.e., connections) between them from SpaceWire to IEEE 1394. After a short time, the *network builder* and all of its children will complete the change. Then, the children nodes send the same request to their own neighbors. However, this time a neighbor node can refuse to become a child if it has already have one edge changed to IEEE 1394, and such a node will not forward the *network building message* either. When a node has no neighbor to pass on the *network building message* or has received refusal from all of its neighbors, then it realizes that it is a leaf node in the IEEE 1394 bus. This approach guarantees that all nodes will be connected by the spanning tree because all nodes will be traversed by the *network building message.* It also guarantees that no loops or circles can be formed because the nodes already joined the tree will not cross-linked with each other.

The maximum time to build the tree can be estimated from the longest path in the connected graph of the SpaceWire. After a node has determined that it is a leaf node, it will wait for the maximum tree building time to pass to ensure that the IEEE 1394 bus has connected all nodes. If there is no bus reset is issued by other nodes during that time, then it will trigger a bus reset. The bus reset will initialize the IEEE 1394 bus. To elect the *network builder,* each system node is provided a special timer and the timeout value of this timer in every node is different. This time starts running at system start up. The node in which the timer times out first will become the *network builder*.

When a node detects an IEEE 1394 bus fault, it will change all of its edges to the SpaceWire. This will eventually force all nodes switching back to the SpaceWire. Since the SpaceWire has very high bandwidth, it can take over most of the spacecraft operations. On the other hand, since its bus latency is more difficult to control in a distributed environment, it has to fall back to a master-slave architecture so that the bus traffic and thus the latency can be easier to control by the bus master. Any node that has detected the IEEE 1394 bus fault can bid for the bus master of the SpaceWire. When a node has completed the change of its connections to the SpaceWire, it will wait for a fixed amount of time. If it does not receive any message from a master node within this time, it will declare to be the master and send out the notice to all other nodes. If more than one node declare to be the bus master, the node with higher ID will win.

After the SpaceWire has re-established, it will isolate the original of the fault, in addition to performing the normal spacecraft operations. When a faulty node or connection is identified, all nodes will remove it from its neighbor list, so that it will not participate in any future network-building process. When the system is ready to rebuild the IEEE bus, the bus master of the SpaceWire can initiate the network building process again.

## 7.3 Effectiveness of the Fault Tolerant Design of the Combined IEEE 1394/SpaceWire Bus Network

If a fault occurs at a branch node or an edge (i.e., connection) of the spanning tree, then the IEEE 1394 bus will be partitioned. In order to recover from the fault, another spanning tree has to be selected from the remaining graph. However, the new spanning tree should not share the failed branch node or edge with the original spanning tree. Otherwise, the new spanning tree will not be able to recover from the fault either. For this reason, there are only a subset of the spanning trees in the graph has useful configurations for fault recovery. The size of this subset is determined by the smallest number of nodes or edges that partition the graph.

**Definition:** A *cut-vertices set* is the smallest number of nodes such that the removal of them partitions the graph.

**Definition:** A *bridge set* is the smallest number of edges that the removal of them partitions the graph.

**Theorem 1:** Given a connected graph G formed by the connections of the SpaceWire, if the size of its cut-vertices set is N and the size of its bridge set is M, then the minimum number of failed nodes or edges that can be tolerated by G is equal to min(N-1,M-1)

**Proof:** Since a spanning tree has to connect all nodes, at least one of its nodes has to belong to the cut-vertices set. Without lost of generality, let's assume the failed node in the spanning tree is a member of the cut-vertices set. In order to recover from the failure, the system has to select another spanning tree from the remaining graph. The new spanning is also required to connect all remain nodes and therefore will contain another member of the cut-vertices set. In the worst case, every time when a tree fails, the node that caused the failure is a member of the cut-vertices set. Since the size of the cut-vertices set is N, in the worst case, the system can only recover N-1 times before the entire connected graph is completely dissected.

The same argument also applies to the bridge set, so that the system can only recover M-1 edge failures before the entire connected graph is dissected. The minimum number of faults that the system can tolerate is the smaller number of N-1 and M-1. *Q.E.D.*

An example of the IEEE 1394/SpaceWire bus network is shown in the following. The thick edges depicts the IEEE 1394 bus connections and the thin edges depicts the SpaceWire.

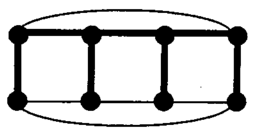


Figure 8: IEEE 1394/SpaceWire Bus Network

This bus network has a minimum cut-vertices set of 4 and the size of bridge set is also 4. Therefore, based on Theorem 1, this bus network is capable of tolerating 3 faults. Comparing to the IEEE 1394/I2C bus architecture, this bus network example has the fault tolerance capability and the same number of connections as a single IEEE 1394 bus in that architecture. However, this bus network does not require I2C bus to support the fault recovery. Also, since the SpaceWire has sufficient performance to temporarily take over the spacecraft workload, there is no need to have a second IEEE 1394 bus to handle normal operations during the recovery of the first IEEE 1394 bus. Therefore, this bus network has the same capabilities as the original IEEE 1394/I2C bus architecture but much simpler connectivity.

## 8. Conclusion

We have desribed a new bus architecture that consists the IEEE 1394 bus and SpaceWire based on the multi-level fault tolerant methodology for COTS. The approach to combine these two buses and the strategy to achieve fault tolerance have been described. It has also been illustrated that, due to the high performance and topological flexibility of the SpaceWire, the new architecture has the same capabilities as the original IEEE 1394/I2C bus architecture but much simpler connectivity.

## 9. Current Status and future work

The ASIC design for the original IEEE 1394/I2C bus architecture is nearly completed and some of the fault tolerance capabilities in the original bus architecture has been demonstrated in a proof-of-concept testbed at the Center for Integrated Space Microsystem (CISM) facility at the Jet Propulsion Laboratory. Hardware of the Spacewire have been procured and experiment of the new bus architecture is expected to begin in early Year 2002.

While the IEEE 1394/SpaceWire architecture has many advantages over the IEEE 1394/I2C architecture, its software is more complicate since it has to switch from a distributive environment of the IEEE 1394 bus to a master-slave environment of the SpaceWire. The complexity and techniques to simplify the software will be examined in future papers.

## Acknowledgement

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

## References

[1] L. Alkalai and A. T. Tai, “Long-life deep-space applications,” *IEEE Computer*, vol. 31, pp. 37–38, Apr. 1998..

[2] IEEE 1394, *Standard for a High Performance Serial Bus*. Institute of Electrical and Electronic Engineers, Jan. 1995.

[3] D. Anderson, *FireWire System Architecture, IEEE 1394*. PC System Architecture Series, MA: Addison Wesley, 1998.

[4] D. Paret and C. Fenger, *The $I^2C$ Bus: From Theory to Practice*. John Wiley, 1997.

[5] Philips Semiconductor, *The $I^2C$-Bus Specification Version 2.0*, Philips Semiconductor, Dec. 1998.

[6] IEEE P1394A, *Standard for a High Performance Serial Bus (Supplement)*, Draft 2.0. Institute of Electrical and Electronic Engineers, Mar. 1998.

[7] W. Peterson, “The VMEbus Handbook: Expanded Third Edition,”, VFEA International Trade Association, 1993

[8] A. Tai, S. Chau, and L. Alkalai, "COTS-Based Fault Tolerance in Deep Space: Qualitative and Quantitative Analyses of A Bus Network Architecture," *Proceedings of the 4th IEEE High-Assurance System Engineering Symposium,* (Washington, D.C.), November 1999.

[9] H. Luong, "X2000 First Delivery Digital Input and Output (DIO) ASIC," *JPL Document D-16931*